4-22-2014

# NLSR: Named Data Link State Routing Protocol

A K M Mahmudul Hoque

# NLSR: NAMED-DATA LINK STATE ROUTING PROTOCOL

by

A K M Mahmudul Hoque

A Thesis
Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

Major: Computer Science

The University of Memphis
May 2014

*To my loving wife*

*and family*

## Acknowledgment

First and foremost, I would like to take the opportunity to thank Allah, the Creator, the Lord of the Worlds, the Beneficent, and the most Merciful, for giving me the courage and ability to complete this task.

I express my sincerest gratitude to my supervisor, Associate Professor Lan Wang, for her constant support, encouragement, inspiration, and guidance, and for having her vote of confidence in me. She has consistently allowed me to pursue my research ideas, provided invaluable feedback to keep me focused and on track, and helped me advance my research. She has been a true mentor, not only in the avenue of research, but also in every other aspect of my life. I would also like to thank Dr. Timothy Hnat and Dr. David Lin for their constructive comments and insightful feedback.

I am truly grateful to have such a wonderful wife and family, who are always inspiring me throughout my life.

With the utmost appreciation, I thank the members of the NDN research group and the members of the University of Memphis Networking Lab, including Syed Obaid Amin, Marc Badrian, Vince Lehman, Ashlesh Gawande, Nic Smith, and Minsheng Zhang. I am also thankful to project collaborators Beichuan Zhang and Lixia Zhang for their added support and invaluable feedback.

Finally, I would like to thank the University of Memphis, for providing me with an excellent research environment as well as supporting me throughout my academic career.

## Abstract

Named Data Networking (NDN) is a fundamental paradigm shift from the current Internet where, packets are forwarded by name instead of the destination IP address. By explicitly naming each packet and signing data, NDN enables some revolutionary features like data authenticity, multicast data delivery, and multipath forwarding with adaptive strategies. For NDN to work well over a network, it requires a routing protocol which will not only need to propagate name reachability in the network, but also compute ranked multipath forwarding entries for each name by ensuring the security of routing exchanges. Moreover, moving from a traditional, long studied, and well-understood IP based thinking process to name based routing makes designing an efficient routing protocol for NDN more challenging. This thesis presents Named-data Link State Routing (NLSR), which propagates name reachability and computes ranked multiple nexthops for forwarding. NLSR also takes advantage of inherent data authenticity features to provide simple yet robust security for routing exchanges.

This thesis focuses on discussing four functional design goals of NLSR. First and foremost is designing a naming scheme for routers, routing updates, and routers' cryptographic certificates. The second design goal is to make a rational choice between two available synchronization protocols for disseminating routing updates in NDN. The third goal is designing an efficient algorithm to produce multiple nexthops for each forwarding entry. The fourth and final goal is to produce a self-sufficient design for naming, distributing cryptographic certificates in the network, and deriving trust from those certificates for routing updates.

The goal of this thesis is to design and evaluate a routing protocol, which will well serve the needs of NDN. NLSR moves from the conventional IP based routing to name based routing and from single path forwarding to multiple path forwarding. We have evaluated NLSR, and compared to IP link state routing protocol, it offers

more efficient routing update propagation, inherent update authentication, and native support of multipath forwarding. NLSR provides a great learning experience to develop an application on top of NDN which requires meticulous consideration in namespace design, careful design of the trust model for data authentication, and most importantly, a mental adjustment to NDN's design philosophy of using interest-data exchanges for routing messages. NLSR is the first distributed routing protocol in NDN for a single authoritative domain and the first step toward developing and extending protocols for inter-domain routing.

# Table of Contents

# List of Tables

# List of Figures

## Citation to Previous Publication

A large portion of this thesis appeared in the following publication:

- A K M M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Wang. "NLSR: Name-data Link State Routing Protocol", In the proceedings of ACM SIGCOMM ICN Workshop, August 2013

# 1 Introduction

The Internet has been evolving since its birth, and communication in the current Internet is dominated by content distribution. To keep up with evolving Internet communication, Internet researchers proposed Named Data Networking (NDN), a new architecture for the future Internet. NDN is a fundamental architectural shift from the current Internet, in which each packet carries a name instead of a destination IP address. Unlike IP, NDN forwards a packet by looking at the names of the packets [5]. Communication in NDN is consumer driven; that is, a consumer issues interest for data by mentioning the name in an interest packet. This interest is forwarded to a producer by looking at the name. A producer replies with a data packet containing the name, data, and the signature of the producer. Upon receiving the data, the consumer can verify the authenticity of the data. Thus, because each packet is named and all data is signed, NDN inherently enables features like data authenticity, in-network caching of data, multicast delivery of packets, and an adaptive forwarding strategy for multipath.

NDN is evolving as the future of the Internet, and for proper network functionality, it needs a routing protocol for generating Forwarding Information Base (FIB) entries and installing them in the forwarding table of a NDN node. Each FIB entry of NDN contains a name prefix and a set of nexthops through which this name prefix can be reached. When any interest comes to an NDN node, the interest's name prefix is matched against the forwarding entries in the forwarding table to forward that interest. Moreover, NDN has an inherent loop prevention

technique [5,6], which allows the use of multiple nexthops for forwarding any packet in the network. So, for efficient and effective adaptive forwarding, NDN needs a routing protocol, that will not only produce name-based forwarding tables but also support multiple nexthops for forwarding entries. Motivated by the need for an effective name based routing protocol for NDN, we present the Named-data Link State Routing (NLSR) protocol which supports name based routing and produces ranked multiple nexthops for forwarding entries. NLSR is built on top of NDN, so every routing update exchange is done in the form: interest/data. Like IP link-state routing protocols such as OSPF [7], NLSR disseminates Link State Advertisement (LSA) updates, builds a network topology from LSA updates, calculates shortest paths, and generates next hops for forwarding entries. However, unlike IP, NLSR produces ranked multiple nexthops for insertion into the forwarding table of an NDN node. Moreover, NLSR takes advantage of data authenticity, which is a built in feature of NDN, to ensure authenticity of routing update exchanges.

The contribution of this thesis is the first distributed routing protocol in NDN, NLSR, which focuses on answering four main design questions: the first and foremost question is how do we design an effective naming scheme to name routers, routing updates, and routers' certificates? In NDN, every entity is identified by name. Although NLSR can use any underlying protocol for routing exchanges, each entity in the protocol needs to be named. So, this is a design challenge in mapping our traditional IP-based cognitive model to NDN's name-based model.

The second design goal is to choose or design an efficient routing updates synchronization protocol to disseminate updates in the network with minimal delay time while also ensuring correctness. Moreover, where in IP routing protocol an update is pushed in the network, the NDN philosophy allows NLSR to pull updates from it. Instead of reinventing the wheel, we evaluated two available synchronization protocols in NDN, NDN-Sync (hop by hop dataset synchronization

protocol) and chronoSync (distributed dataset synchronizaition protocl), in order to find a functionally correct, optimal, and well suited for NLSR.

The third goal of NLSR design is to create an algorithm which produces ranked multiple nexthops to facilitate multipath forwarding thereby taking advantage of the adaptive forwarding strategy in NDN [15]. While IP routing produces single best nexthops or limits its forwarding to equal cost multipath, NDN routing needs to produce multiple nexthops for each forwarding entry to make the best use of NDN's intelligent and adaptive forwarding strategy.

The fourth and final design goal is to exploit NDN's built in data authenticity features to ensure security of routing updates without exposing them to any potential security threats. NLSR is directly benefitted from the built in feature of data authenticity in NDN. As routing updates are NDN packets, which carry name, data, and a signature, a receiving router can verify the signature of routing update packets. The router can then ensure that the routing update packet was generated by an authentic router and was not tampered with during the dissemination process. The contribution of this thesis towards achieving this design goal is finding a secured distribution process for cryptographic certificates as well a method to derive trust from these certificates.

This thesis describes the design choices and rationale to achieve these four main design goals in NLSR. The goal of this thesis is to present the feasibility and benefits of using a link state routing protocol in NDN rather than inventing one. We evaluated NLSR, and compared to IP link state routing protocol it offers more efficient routing update dissemination, built-in update authentication, and native support for multipath forwarding. The remainder of this thesis is organized as follows: Chapter 2 provides brief background knowledge on Named Data Networking (NDN) and link state routing. Chapter 3 describes design details of our research work. Chapter 4 presents the evaluation of the NLSR protocol. Chapter 5

discusses related works in NDN routing area, chapter 6 presents planned future works, and finally, chapter 7 concludes the thesis.

# 2 Background

NDN is an architectural shift from today's IP, in which the concept of *"where"* in IP is replaced with concept of *"what"*. Instead of forwarding packets to *where* – the destination addresses, NDN forwards packets to *what* – the content that users in today's content-driven Internet really care about. In this chapter, we provide the background on Named Data Networking (NDN) architecture. We also provide a brief background on Link State Routing protocol at the end of this chapter.

## 2.1 Named Data Networking

As a result of its continuous evolution, today's Internet has reached a point where most of its communication has become content-driven, but the current IP communication model was designed to be destination-driven. To better suit today's content-driven Internet communication pattern, Jacobson et al. proposed a new Internet architecture: Named Data Networking (NDN) [5]. In this section, we will briefly discuss the basic concepts of NDN, which is the foundation for our work.

### 2.1.1 NDN Packets

All packets in NDN are of two types: *interest* and *data* as depicted in Figure 2.1. *Interest* packets are sent when consumers express interest for *data*. Each *interest* packet carries a name for the desired *data*, a selector field indicating preferences and restrictions if multiple content matches are found in response to the interest, and a nonce (a random value used to detect duplicates). *Data* packets contain a name which is used to match *interest* of consumers), the data content, the publisher's

signature, and additional signing information about the publisher's cryptographic certificates, validity, etc.

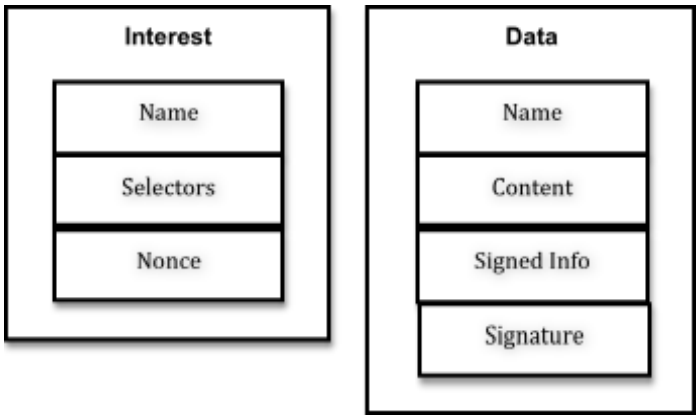


Figure 2.1: Two types of NDN packets:*interest* and *data*

### 2.1.2 Hierarchical name

Each and every packet in NDN is identified with a hierarchically-structured name, which is unique under a scope. For example, this thesis can be named as `/memphis.edu/cs/ahoque/thesis/msthesis.pdf`, where each '/' represents a boundary between name components and is not part of the name. Naming conventions vary from application to application and are opaque to the network [6]. Because of the name's opacity, applications can choose any naming scheme and evolve from the network independently. Furthermore, this hierarchical naming structure enables applications to represent a relationship between two pieces of data.

### 2.1.3 NDN node

Each NDN node has three main components in its forwarding plane [5]: the Forwarding Information Base (FIB), the Pending Interest Table (PIT), and the Content Store (CS). The FIB stores forwarding entries used to forward *interests*

towards a potential matching *data* source(s). Unlike IP, NDN allows a list of outgoing faces rather than a single, best next-hop or equally costly multi-hops. The PIT stores the unsatisfied *interest* names along with the faces they are received from so that *data* packets can be routed back to the consumers by following the *interest* trail. The CS is used for caching *data* in a NDN node. Figure 2.2 shows the forwarding plane model of a NDN node taken from [5].

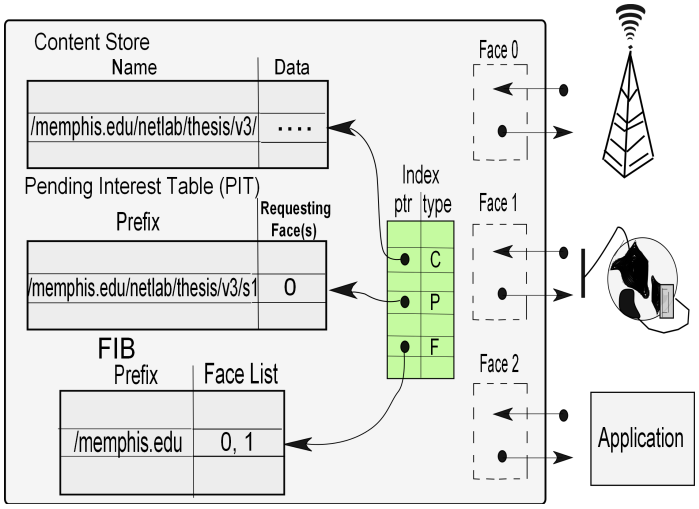


Figure 2.2: Forwarding plane of NDN Node consists of Forwarding Information Base (FIB), Content Store (CS), and Pending Interest Table (PIT) [5]

### 2.1.4 Consumer-driven communication

Communication in NDN is consumer-driven, where the consumer initiates communication by expressing an *interest* for desired *data*. A NDN node forwards this *interest* to the producer(s) of the *data* according to the entries in the FIB, which is populated by a name-based routing protocol. The node also remembers the name of this *interest* and adds the incoming face as an entry in the PIT. When the *interest* packet reaches a *data* packet matching its name, the *data* packet traverses back toward the consumer by following a reverse path laid out by the *interest*. All

unsatisfied *interests* are recorded in the PIT. If multiple *interests* exist for the same name, a NDN node does not forward this *interest* again upstream since there will already be a PIT entry matching this name. Instead, another incoming face will be added to the PIT entry's face list. Later on when the corresponding *data* arrives, the node checks the PIT for the unsatisfied matching *interest*, finds the list of faces, and sends the *data* out to all those faces downstream. So NDN's forwarding plane supports multicast packet delivery, as shown in Figure 2.3.

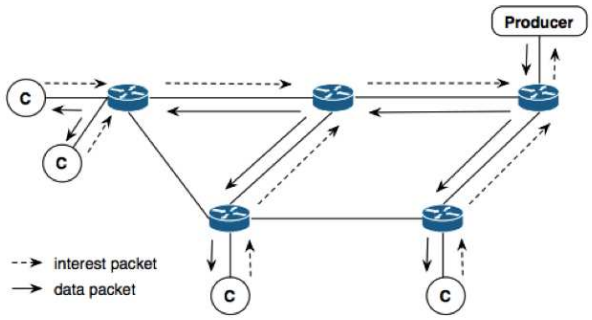


Figure 2.3: Consumer-driven communication supporting multicast delivery in NDN [16]

### 2.1.5 Intelligent forwarding plane

A NDN node preserves the state of each *interest* in the PIT after forwarding it. So each entry in the PIT indicates unsatisfied *interest* from the downstream node and is removed if matching *data* satisfies the *interest*. This per-packet state information not only allows NDN to forward packets to multiple faces and avoid loops, along with the *data* returning success feedback it also enables NDN to be adaptive in reaction to network failures. This ultimately leads to first alternative path exploration and better network resource utilization. Moreover, a NDN node can also limit the rate of incoming packets by limiting the PIT size [6].

### 2.1.6 Data centric security

NDN secures *data* packet instead of securing communication channels [4, 5]. Each *data* packet contains the publisher's signature, coupled with its name. The signature, along with the publisher's information, allows consumers to authenticate the *data*, verify its provenance, and derive trust from a model. This security approach makes data independent of where it comes from, allowing NDN to cache *data* anywhere in the network to satisfy any future interest. More importantly, by leveraging this end-to-end security approach of NDN, consumers and publishers of data (applications) can tailor and customize their own trust model.

## 2.2 Link State Routing

Link state routing is a global routing scheme where each router originates Link State Advertisements (LSAs) about itself, and its connectivity states. These LSAs are disseminated in the network from router to router. After network convergence, every router in the network has a map of the network, which is used to independently compute individual best paths to a reachable destination. The basic functionality and concepts of link state routing are discussed briefly in this section.

### 2.2.1 Building Adjacency

Each router maintains a relationship called *'Adjacency'* with its neighbors after being discovered via a Hello protocol [7]. Hello packets from the Hello protocol also serve the purpose of monitoring the adjacency status of neighbors. This adjacency relationship plays an important role in the Link State Advertisement synchronization process.

### 2.2.2 Link State Advertisement (LSA) synchronization

After establishing adjacency with its neighbor, the router builds LSAs by encompassing its connectivity and state information and sends them out to its adjacent neighbors using a reliable flooding technique. Each router receiving the

LSAs first copies them into its own Link State Database (LSDB), then sends them out again to that router's adjacent neighbors (except for the incoming one) in an area. Thus each and every router advertises and gathers knowledge of the network. Aging and sequencing is applied to expire old LSAs and replace them by propagating new LSAs respectively.

### 2.2.3 Routing table calculation

Following network convergence, every router's LSDB should be identical in a area. At this point, every router in a area has the same complete picture of the network in order to compute the shortest path to all reachable destinations. Then each router builds a network topology from the Link State Database, considering itself as the root of the network tree, and applies Dijsktra's shortest path algorithm to calculate the shortest path to all destinations. After shortest path calculation, each router generates its own routing table by calculating next-hop to all reachable destination.

# 3 Design and Implementation

NLSR is a link state routing protocol for NDN. Like other link state routing protocols (OSPF [7], IS-IS [8]), NLSR propagates LSAs for building the network topology as well as distributes name prefix reachability. NLSR discovers, establishes, and maintains adjacency with neighboring routers. Whenever NLSR detects any changes in adjacency's state such as link failure, link recovery, neighbor router crashing, or neighbor router recovery, it updates the LSA and disseminates the new LSA to the network. NLSR also advertises name prefixes from both static configuration and dynamic registration by content producers. Upon every deletion or addition of a name prefix, NLSR disseminates a new LSA. NLSR's LSDB always keeps the latest version of the LSAs.

NLSR's such dissemination of LSAs and building the topology may first appear to be very trivial, as identical functionalities have already been implemented in IP routing protocol. However, since NLSR is implemented on top of NDN, it needs to use Interest and Data packets, and the design must shift away from the familiar concepts of destination IP addresses to *name* prefixes and from data pushing into the network (where any node can simply send any packet to any other node) to data pulling from it. Therefore, in NDN we have to think in terms of packet names and data retrieval. To be more precise, we need a methodical naming system for routers, routing updates (Section 3.1), and router's cryptographic certificates (Section 3.6). Moreover, we need to pull routing updates promptly and without previous knowledge of when an update may be generated (Section 3.3).

In terms of routing functionality, NLSR distinguishes itself from all other link state routing protocols by two facets: it produces multiple routes for each name prefix in place of single best next-hops, and it signs and verifies all routing updates messages to ensure that each router can originate and disseminate only its own prefix and connectivity information within the network. We present our route calculation algorithm in Section 3.4 and our trust model in Section 3.6. As a first step in developing this NDN-based routing protocol, our initial design of NLSR lies in the context of a single routing domain with a single authority on which our trust model is built. We truly believe that this design and deployment experience of NLSR in the NDN research test bed can offer us a concrete stepping stone towards developing an inter-domain routing protocol that incorporates routing policies and an inter-domain trust model.

## 3.1 Naming

Designing a functional naming scheme for each element: routers, routing updates in the routing system, and the router's corresponding cryptographic certificates is perhaps the most challenging and important element of NLSR's design. Based on current operational practices and network structuring schemes, a hierarchical naming system best suits capturing the relationships among various network components in the system. This makes it easy to identify routers belonging to the same network, sites, and messages generated by a specific router.

Every router in our design is named by following a structured hierarchical naming scheme. The first part of the router name is the network this router belongs to, the second part is the site owning the router, and the final part is the assigned router name. So a router name becomes of the form `/<network>/<site>/<router>`. For instance, an ATT router in Atlanta PoP (point of presence) may be named `/ATT/AtlantaPoP/router7`. This way we know that if two routers share the

same /<network> prefix then they belong to same network, and if they share the same /<network>/<site> prefix, then they belong to same site. This naming scheme not only makes it easy to filter out erroneous, unwanted routing messages, but it also derives trust (Section 3.6).

The NLSR process on a router is denoted by the process name following the router name: the router name is used as its prefix, followed by the process name that NLSR constitutes to the form/<network>/<site>/<router>/nlsr. This process name is used in periodic info message exhanges between neighboring routers for establishing adjacency, detecting failure, or the recovery of either links or routing processes (Section 3.5). Moreover, this process naming scheme of process provides the scope of running other routing processes on the same router in the distant future.

Routing updates are named with the prefix /<network>/nlsr/LSA/<site>/<router>. Ideally, any updates originated by a NLSR process should have the process prefix /<network>/<site>/<router>/nlsr/LSA, indicating that it has been generated by a NLSR process of router /<network>/<site>/<router>. However, since our implementation uses ChronoSync [16] to disseminate LSA data in the network, all routing updates need to share a common routable prefix in their name. ChronoSync does not impose any constraints on the naming of routing updates however, the fact that it assumes that all data names to be synched are routable creates a circular reference to the routing process itself. To avoid this circular indirection, all routing updates generated by a NLSR process share a common prefix /<network>/nlsr/LSA(we call this the <LSA-Prefix>), and append /<site>/<router> at the end to differentiate LSAs originated by different NLSR routers.

## 3.2 LSAs

NLSR originates and disseminates three types of LSAs: a) Adjacency LSAs, b) Prefix LSAs, and c) Hyperbolic LSAs. All the LSAs of NLSR have the name format: `/<LSA-Prefix>/<site>/<router>/<LSA-type>/<sequence-no>`, where `<router>` is the name of the router that originates the LSA, `<LSA-type>` is the type of LSA (Prefix, Adjacency or Hyperbolic LSAs), and `<sequence-no>` is an integer used to determine the ordering of a particular LSA as it changes over time. The Adjacency LSA is used to advertise all active adjacency with neighboring NDN routers. Each adjacency link description of an Adjacency LSA contains a neighboring router name associated with a cost to reach that neighbor. NLSR builds an adjacency list at startup time and creates the Adjacency LSA. Any changes in the status of any link to a neighbor or neighboring process detected by periodic "info" Interest messages (Section 3.5) triggers updates by the propagation of the Adjacency LSA.

Table 3.1: Contents of an LSA

| **LSA Type** | **Content** |
|---|---|
| Adjacency LSA | # Active Links (N), Neighbor 1 Name, Link 1 Cost, ..., Neighbor N Name, Link N Cost |
| Prefix LSA | Name Prefix |
| Hyperbolic LSA | Hyperbolic Radius, Hyperbolic Cordinate |

The Prefix LSA advertises all name prefixes (statically configured and dynamically advertised by application) that are reachable from the origination router. Withdrawal or addition of any name prefix causes a NLSR router to update and disseminate the Prefix LSA thoughout the network. Hyperbolic LSAs contain the geometric hyperbolic coordinates of the origination router, which are used for hyperbolic routing table calculation [9].

In order to remove outdated LSAs caused by router crashes and network partitions, every router periodically refreshes each of the advertised LSAs by

generating a newer version with an incremented sequence number. Each LSA has a lifetime associated with it and is removed from the LSDB when its lifetime expires. Therefore, if a router crashes, its LSAs will not persist in the network. If a network is partitioned because of periodical LSA expiration, the LSAs of one partition will be discarded from another partition to ensure consistency in the network. One important thing to note here: route calculation should not be impacted by the obsolete LSAs in NLSR. If a router crashes or a link goes down, its neighbors will update the status of their LSAs and propagate them throughout the network, so traffic will not be directed towards those routers or by those links. Since we do not use the refreshes to handle packet losses or state corruption (Sync Protocol handles that), and the obsolete LSAs do not affect routing table calculations, the LSA refreshing interval can be set to a relatively long period.

## 3.3 LSDB Synchronization

For the conceptual simplification of our design, we view the LSDB as a collection of *data* and the LSDB synchronization problem as a *data* synchronization problem of the LSDB as maintained by the routers. Routers periodically exchange their hashes of the LSDB to detect inconsistencies and recover from them. This synchronization approach avoids unnecessary flooding to the network – when the network is stable, as only one hash (instead of all the LSAs) is exchanged between neighbors. Moreover, it is also receiver-driven, implying that a router will request LSAs only when it has CPU cycles available.

### 3.3.1 NDN Synchronization protocol (Sync)

In the early stages of NLSR implementation, NLSR uses the NDN synchronization protocol [10] to disseminate LSAs to neighboring routers. Sync is associated with the NDN repository, which allows applications to define collections of named *data* called slices in repo, which are then kept in sync with identically defined slices in

neighboring repos. Sync computes a hash tree over all the data in a slice and exchanges the root hash between neighbors to detect discrepancies. If the hash values do not agree, two neighboring nodes then exchange the hash values with nodes on the next tree level continuously until they detect the specific leaf node(s) causing the problem. They then exchange the data to reach consistency.

Figure 3.1 shows how a LSA is disseminated in the network. To synchronize the slice containing LSAs, the Sync protocol periodically sends special Interest messages called Root Advise messages, along with the hash value of the slice, to the neighboring nodes (step 1). When Router A's NLSR creates a LSA and writes it in the Sync slice (step 2), its hash value becomes different from that of Router B. This causes Router A's Sync to reply to the Root Advise Interest from Router B with the new hash value of its local slice (step 3). Router B's Sync then compares the hashes and recursively requests for the next level of hashes that are causing the differences. Eventually, Router B's Sync identifies the data that needs to be synchronized (LSAs in the context of NLSR) and retrieves them using Interest messages (step 4 and 5). The Sync on Router B then sends the data name to the local NLSR agent (step 6), which fetches the data from the local repo (step 7 and 8) and updates its LSDB (step 9).

Although the NDN synchronization protocol gives NLSR an efficient hop-by-hop dissemination of LSAs in network, it fails to notify NLSR in case of frequent updates in a large network, making NLSR functionally incorrect. Moreover, NDN synchronization retains all old copies of data, causing memory usage to grow with each LSA refresh interval by the size of all LSAs in the network, which makes NLSR unfeasible to run for a relatively longer period.

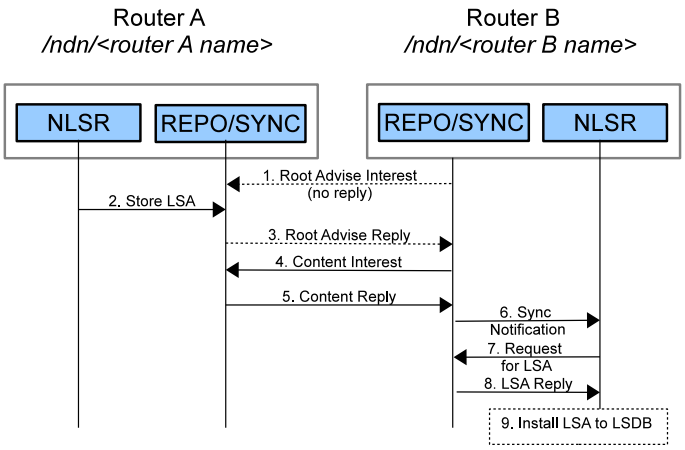


Figure 3.1: LSA dissemination process from Router A to Router B via NDN synchronization protocol

### 3.3.2 ChronoSync

ChronoSync takes a distributed approach instead of a hop-by-hop approach like the NDN synchronization protocol. ChronoSync forms a cryptographic digest by summarizing the state of the dataset of the all-involved parties (the router in our case) in the network, and exchanges it among them. Each entity can detect differences in the dataset from the digest and can decide which data to fetch, whether to fetch, and when to fetch [16].

ChronoSync is implemented as a library, and any application can incorporate it for dataset synchronization. When there is a data update, ChronoSync increases the sequence number associated with the data name, computes the digest, and exchanges the digest with others to fetch the updated data name(s). If there is an update in any data name, the application can detect it and decide what to do with that update. However, NLSR itself needs to increase the sequence number of the LSAs when generating a LSA update instead of letting ChronoSync do it. Therefore we modified the ChronoSync library so that it can adapt with the NLSR requirements of pushing the sequence number with the LSA updates.

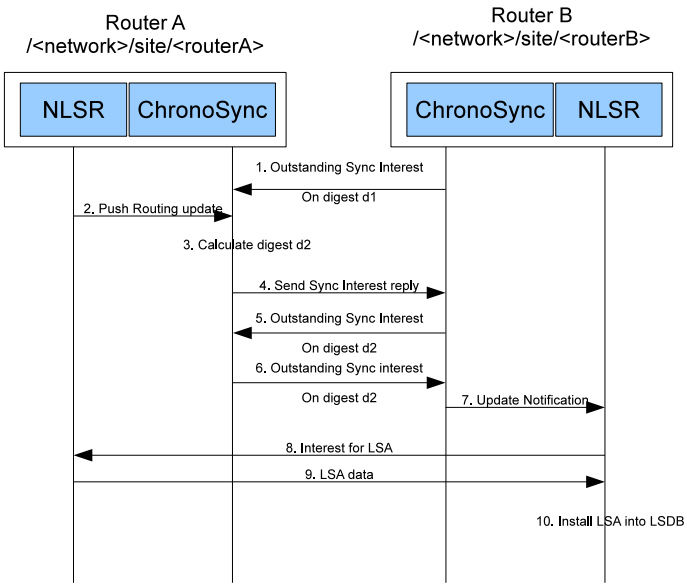


Figure 3.2: LSA dissemination process from router A to Router B via ChronoSync

Figure 3.2 shows how NLSR disseminates LSAs via ChronoSync. Lets assume that at any stable state both routers have outstanding sync interest on digest *d1* (step 1). Router A updates its LSAs and pushes updates to ChronoSync in step 2. In step 3, ChronoSync calculates digest *d2* and sends sync data for outstanding sync interest on d1. ChronoSync on router B calculates its own digest *d2* and both routers send outstanding interest for fetching future updates (step 5 and 6). In step 7, ChronoSync on router B notifies the NLSR process about the routing update. Router B then sends interest for the updated LSA to router A in step 8. In step 9, router A replies back with LSA data in response to step 8's interest. In step 10, router B installs the LSA into the LSDB.

ChronoSync's ability to handle simultaneous data generation without failing to notify applications and to recover from network partitions makes it an ideal candidate for routing updates dissemination in order to ensure correctness. Unlike NDN Sync, ChronoSync does not store any content, which makes it more feasible

for long-term usage with any application. Because of its correctness and all the other benefits over NDN synchronization protocol, we chose to use ChronoSync for LSA dissemination with NLSR.

## 3.4 Multipath Calculation

Each NLSR node constructs a network topology using the information from the Adjacency LSAs in the LSDB. Afterwards, it runs a simple extension of the Dijsktra's algorithm to produce multiple next-hops for each destination node. But if a router is configured to calculate the routing table using hyperbolic information, NLSR calculates the routing table according to the hyperbolic routing algorithm presented by Fragkiskos et al [9]. From the Prefix LSAs, we know which name prefix is originated from which destination router. So from prefix LSA we get destination router for name prefix, and from routing table we obtain list of next-hops to reach that destination. By combining these two information, we can obtain a list of next-hops to reach each name prefix.

Our multipath calculation using Dijsktra's algorithm works as follows: It removes all immediately adjacent links except one, then uses Dijkstra's algorithm to calculate the cost of using that link to reach every destination in the topology. This process is repeated for every adjacent link. In the case of hyperbolic routing calculation it's trivial, as the algorithm checks the distance from all neighbors to the destination and multipath calculation is done in one pass. Afterwards, NLSR ranks the next-hops for each destination based on the route cost to reach that destination. Note that NLSR allows the operator to specify the maximum number of paths per name prefix to insert into the FIB, so that the FIB size can be controlled when a node has many neighbors. However, the computational cost still increases as the number of neighbors increases, because the algorithm checks through all the neighboring links to produce the cost for each path and then ranks them.

Unlike IP, routing information in NDN acts only as a hint to the forwarding plane. The forwarding plane can observe data delivery performance using state information maintained in the PIT and then rank the multiple next-hops of a name prefix using the actual observation as well as the ranking from the routing protocol [15]. However, the ranking information from the routing protocol is still important in forwarding the initial *interest* to a name prefix and in exploring alternative routes when the current route fails to retrieve data.

## 3.5 Failures and Recovery Detection

NLSR sends out periodic *"info" interest* messages through each link for detecting link failures and/or remote NLSR process failures. If an *"info" interest* times out, NLSR will try re-sending it at specific times with short intervals to make sure the interest is not lost. If no response to the *"info" interest* is received from the neighbor at the other end, adjacency with that neighbor is considered down. Later on, NLSR continues to send these periodic *"info" interests* to detect the recovery of this adjacency, but at a relatively long interval to avoid high message overhead during a long-lasting failure. For NLSR processes, it is impossible to determine whether the remote NLSR process has died or the connecting link has failed. However, this distinction is insignificant since in both cases the link should not be used to forward any kind of traffic.

When any failed link recovers or a dead remote NLSR process comes alive, NLSR will receive a response to the *"info"* Interest and change that adjacency status to *'Active'*. Any change in adjacency status due to failure/recovery of either a link or a NLSR process results in updating the Adjacency LSAs, disseminating the LSA throughout the network, and scheduling routing table calculation. Figure 3.3 illustrates how Node A detects an adjacency failure with Node C and recovery with

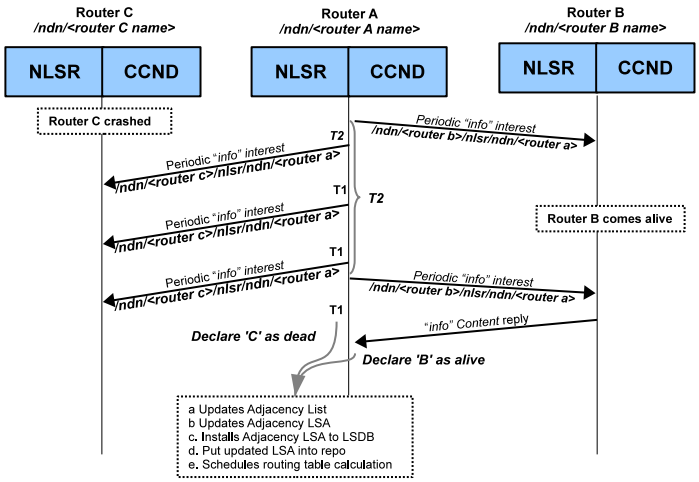


Figure 3.3: Router A's adjacency failure detection of Router C and adjacency recovery detection of Router B

Node B.

## 3.6 Security

Each and every packet in NDN is digitally signed, and the signature is attached as a part of the packet. The signature covers the content, binds the name with that content, and includes a small amount of signature information useful in data verification [5]. One piece of the supporting data is the key locator [1, 5], which indicates the name of the key used to sign the packet so that the receiver can fetch the certificates to verify the signature.

A LSA with a valid signature states that the signature was produced using the public key indicated in the key locator field but it does not verify the provenance of the LSA. For an instance, any attacker can sign a Prefix LSA with his key and inject the LSA into the network. So to check the authenticity of the *data*, a process not only needs to verify the signature, but it also must authenticate that this LSA is indeed signed by an authorized NLSR process. In other words, we need to check that the key has the correct name of the corresponding NLSR process. This still

does not eliminate the possibility of an attacker forging a key with the same name. We then need a trust model to verify the authenticity of the key and a secured system for key distribution and revocation.

### 3.6.1 Trust Model

NLSR is an intra-domain routing protocol. In the context of a single network domain, there is usually a network administrator (a trust anchor) that can certify the authenticity of keys in the network. Therefore we use this trust anchor for key signing and verification, which is easy to setup and manage. We could let this trust anchor sign the public key of every router, but this approach presents a greater security risk when one key is used to sign a large number of keys. Instead, we design a hierarchy of five levels that is rooted at the trust anchor, which limits the signing scope of each key to a smaller size. Table 3.2 shows the name of each key at every level of the hierarchy. Note that the last component of a key name is always the hash of the key (not shown in the table), so that when someone expresses an Interest to a key, the name always matches a specific key. At the top level of the hierarchy is a root key owned by the network domain's administrator. The next level is a set of site keys, each owned by the administrator of a single site in the domain (where a site can be a department in an organization or a PoP in an ISP), that are signed by the root key. Each site key signs a set of operator keys (there may be more than one operator for a site). Each operator key signs a set of router keys, each of which signs the key of the NLSR routing process on that router. Finally, the NLSR key signs the routing data originated by the NLSR process.

Table 3.2: Keys Names

| **Key Owner** | **Key Name** |
|---|---|
| Root | `/<network>/keys` |
| Site | `/<network>/keys/<site>` |
| Operator | `/<network>/keys/<site>/O.Start/<operator>` |
| Router | `/<network>/keys/<site>/R.Start/<router>` |
| NLSR | `/<network>/keys/<site>/.R.Start/<router>/nlsr` |

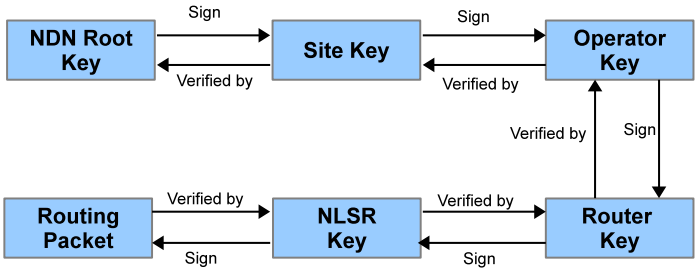


Figure 3.4: Signing and verification process of each routing packet

NLSR strictly enforces the trust model rooted at the trust anchor. Figure 3.4 depicts the flow of signing and the verification process of each NLSR packet. When a NLSR router sends a LSA to the network, it signs the packet with its NLSR key and puts the key name in the *'SignedInfo/KeyLocator/KeyName'* field of the Data packet. Upon receiving a LSA, a NLSR router fetches the key from its certificate store or sends an Interest to fetch the key to complete the verification process. NLSR also checks whether the key indeed belongs to the origination router's NLSR process. This process repeats until NLSR reaches the self-signed key of the trust anchor. If at any step key fetching is unsuccessful, NLSR finds that an unauthorized key signed the key, or the final verification step does not reach the trust anchor, the LSA is considered illegitimate and its discarded as being unsolicited. Note that once a key is verified, we record this information and do not repeat the verification on this key for future packets.

### 3.6.2 Key Distribution

NLSR distributes keys in the network using ChronoSync during startup time. Each router publishes an update in the name of its certificate with a new sequence number. Other routers receiving these update messages from ChronoSync fetch these certificates and try to verify each certificate by looking at the certificate

names included in the signed information (Section 3.6.1). If signer certificates are available in the certificate store and the router can verify the certificate, it is stored in certificate store. Otherwise, the router sends an interest to fetch the signer certificate. This process continues until the router reaches its trust anchor. Certificates are discarded for the same reason a data packet is discarded as described in the previous section. Associating a sequence number with certificates also allows NLSR to revoke signing keys and certificates. The router needs to publish updates of certificate names with a new greater sequence number. Other routers will fetch this certificate and replace older certificate with the newer certificate after successful validation by NLSR.

## 3.7 Implementation

Figure 3.5 shows the block diagram of modular NLSR implementation. Mainly, NLSR can be divided into four components: a) LSDB b) Routing c) Security and d) Communication. The LSDB stores all LSAs (Section 3.2) from the network during the LSDB Synchronization process described in Section 3.3. The routing module builds a network topology by gathering knowledge from the LSDB, it calculates the routing table (Section 3.4), and it generates the FIB for NDN. The security module implements the trust model, stores certificates, signs outgoing data packet, and authenticates incoming data packets before any processing (Section 3.6).

The communication module consists of three sub-modules: i) Data Manager, ii) Interest Manager, and iii) Sync Logic Handler (SLH). The Interest Manager is responsible for sending out and receiving interest. The received interest is forwarded to the Data Manager for further processing. The Data Manager takes care of sending out data in response to incoming interest, and it processes incoming data from neighbors. The Data Manager communicates with the security module to sign

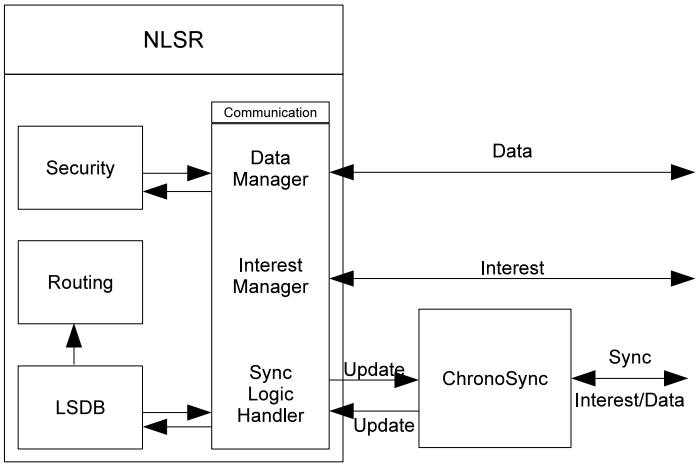


Figure 3.5: Block diagram of NLSR modules

data before sending it out and to validate any incoming data packet. The SLH is responsible for publishing LSA updates in the network and for notifying NLSR about updates from the network. The LSDB module informs SLH about LSA updates, incorporating LSA names with the latest sequence number. The SLH publishes updates to the attached ChronoSync. ChronoSync then synchronizes updates across the network. When ChronoSync detects any updates from network, it immediately notifies the SLH. The SLH communicates with the Interest Manager to fetch this update. The inbound/outbound arrows attached to the Interest Manager and Data Manager connects them to the neighboring NLSR process, and the arrow from ChronoSync connects it to the neighboring ChronoSync process.

# 4 Evaluation

In this chapter, we evaluate the performance of NLSR in terms of processing time, messaging overhead, and convergence time. For conducting tests, we built a network consisting of six heterogeneous nodes with different operating systems and specifications. Figure 4.1 shows the topology we used to represent these results. Although refresh time for LSAs can be set at long intervals (Section 3.2), in order to test the protocol in a short period of time we set the refresh time to be every 30 minutes instead of on the order of days. This allowed us to carefully observe all functionality and measures of performance in a short period of time.

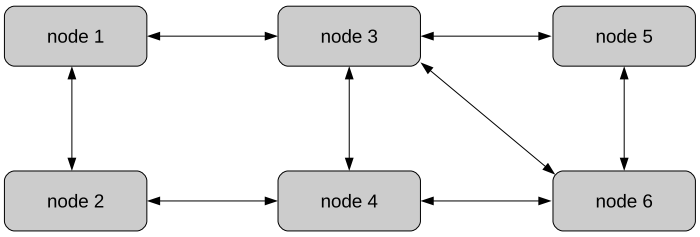


Figure 4.1: Network Topology

Figure 4.2 represents the CPU usage of the NLSR process at each node. The number in parenthesis following the node name indicates the degree of the node (how many active neighbors that are connected to the node). It is evident from Figure 4.2 that the nodes with a higher degree of connectivity in the network exhibit higher CPU usage, which means the computational or processing cost

increases linearly with the number of active links of nodes. This behavior is exhibited by NLSR mainly because of the per-link shortest path calculation (section 3.4), and a higher message-exchanging overhead.

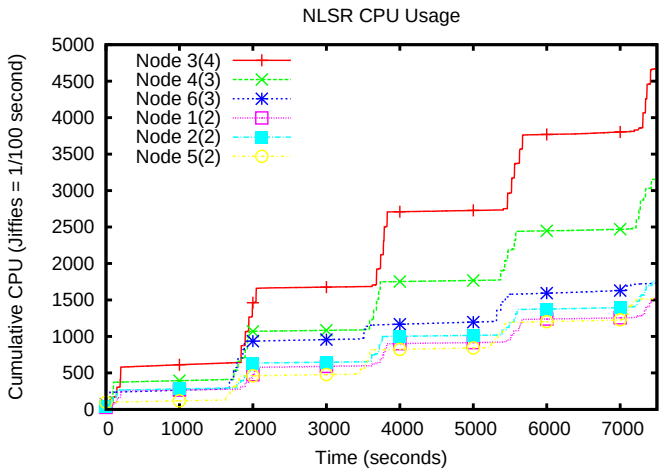


Figure 4.2: NLSR CPU utilization increases proportinately with the number of active link

Figure 4.3 demonstrates the comparison of NLSR's processing overhead with and without the proposed trust model. It is notable that even with the proposed trust model, which requires multiple levels of keys to sign and verify a packet, the extra processing cost is hardly significant. This is due to the fact that NDN by default signs all outgoing data packets. The only noticeable difference between the two schemes is the verification process. NLSR, with the proposed trust system, distributes keys at the startup, which increases the probability of having a high certificate store hit ratio during the verification process. NLSR only requires fetching new certificates; it stores the certificates once after verification, resulting in a very low CPU cost. Figure 4.3 further illustrates the higher CPU usage of multipath routing calculation as opposed to single path routing calculation. Since

the CPU cost due to messaging is the same in both schemes, the difference here is mainly due to the higher processing cost of multipath calculation.

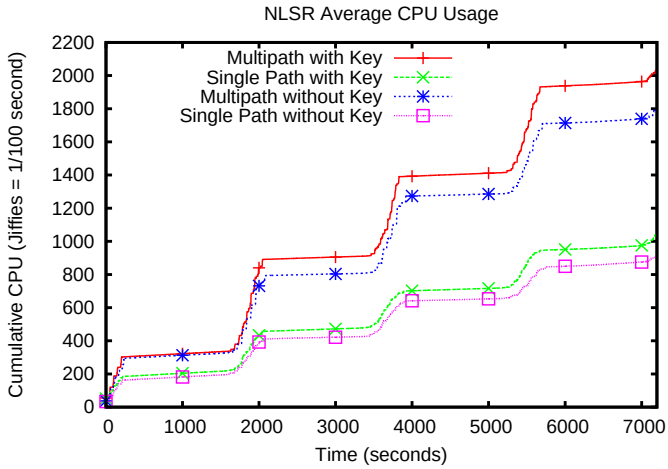


Figure 4.3: NLSR's Average CPU utilization: even with key management NLSR has almost same CPU utilization for both multi and single path

For the convergence test, we used the same topology shown in the Figure 4.1. After booting up all the nodes, we waited long enough to let all the LSDBs synchronize. Once the network was in a converged state, we generated traffic using the ccnping utility [13]. We hosted the ccnping server on node 6, while node 2 was used to generate ccnping ping messages (Interest) with a default timeout value of 4 seconds. After 60 seconds we brought down node 4, which forces the ping messages from node 2 to take an alternative path to reach node 6. Figure 4.4 shows the benefits of multipath routing, wherein node 2 did not need to recalculate the path again. Instead, traffic moved to an alternate path as soon as the failure was detected. NLSR with single path calculation, however, took more than a minute to find the alternative route and moved back to old router as soon as we restored the link. Note that the convergence time can be fine-tuned by tweaking the values of

the *"info"* interest interval, its time out value, and the *"info" interest*'s retry number (which were set to 60 seconds, 15 seconds and 3 times respectively for this test). With these values and configuration settings, link failure detection can take anywhere from 45-105 seconds. By reducing the values, NLSR convergence time can be reduced, but it will result in increasing the number of routing messages throughout the network.

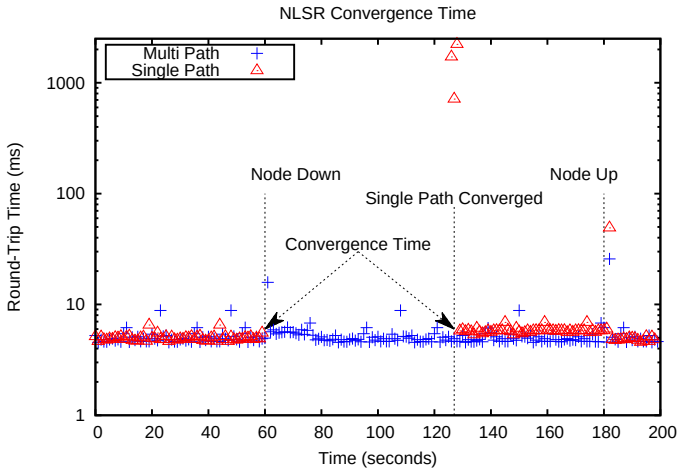


Figure 4.4: NLSR convergence time with & without multipath support. With multipath support traffic takes alternative path immediately(blue), but single path takes about 60 seconds to converge

Table 4.1 shows the number of messages exchanged by OSPFN (Section 5). Table 4.2 demonstrates the number of messages exchanged by NLSR. NLSR exchanges slightly more messages (11.8 messages per link per minute) than compared to OSPFN (10.2 messages per link per minute). With the current implementation, each NLSR process sends interest messages to fetch data when it gets a notification for an LSA update, which results in extra message exchanges for interest and data. We can reduce these LSA interest and data exchanges by piggy

Table 4.1: OSPFN Message Count

| | Links | Hello | LS Update | LS Ack | Total/Node |
|---|---|---|---|---|---|
| **Node 1** | 2 | 1446 | 728 | 296 | 2470 |
| **Node 2** | 2 | 1441 | 884 | 161 | 2486 |
| **Node 3** | 4 | 2886 | 1250 | 739 | 4875 |
| **Node 4** | 3 | 2166 | 1040 | 457 | 3663 |
| **Node 5** | 2 | 1442 | 697 | 321 | 2460 |
| **Node 6** | 3 | 2165 | 1235 | 324 | 3724 |
| **Total/Type** | 16 | 11546 | 5834 | 2298 | 19678 |
| **Avg/Node** | 2.7 | 1924.3 | 972.3 | 383 | 3279.7 |
| **Avg/Link** | 1 | 721.6 | 364.6 | 143.6 | 1229.9 |
| **Avg/Link/Min** | | 6 | 3 | 1.2 | 10.2 |

backing the updated LSA data with sync data. This improvement will reduce the number of messages exchanged by NLSR to 9.04 per link per minute (presented in table 4.3), which is lower than the number of messages exchanged by OSPFN

Table 4.2: NLSR Message Count

| | Links | Info Interest | Info Data | Sync Interest | Sync Data | LSA Interest | LSA Data | Total /Node |
|---|---|---|---|---|---|---|---|---|
| **Node 1** | 2 | 242 | 241 | 1204 | 534 | 426 | 278 | 2925 |
| **Node 2** | 2 | 242 | 241 | 1237 | 419 | 386 | 235 | 2760 |
| **Node 3** | 4 | 484 | 484 | 2199 | 1413 | 805 | 627 | 6012 |
| **Node 4** | 3 | 363 | 363 | 1855 | 686 | 671 | 300 | 4238 |
| **Node 5** | 2 | 242 | 242 | 1141 | 365 | 451 | 147 | 2588 |
| **Node 6** | 3 | 363 | 363 | 1770 | 667 | 633 | 431 | 4227 |
| **Total/Type** | 16 | 1936 | 1934 | 9406 | 4084 | 3372 | 2018 | 22750 |
| **Avg/Node** | 2.7 | 322.7 | 322.3 | 1567.7 | 680.7 | 562 | 336.3 | 3791.7 |
| **Avg/Link** | 1 | 121 | 120.9 | 587.9 | 255.3 | 210.8 | 126.1 | 1421.9 |
| **Avg/Link/Min** | | 1 | 1 | 4.9 | 2.1 | 1.8 | 1.1 | 11.8 |

Table 4.3: NLSR Message Count

| | Links | Info Interest | Info Data | Sync Interest | Sync Data | Total /Node |
|---|---|---|---|---|---|---|
| **Node 1** | 2 | 242 | 241 | 1204 | 534 | 2221 |
| **Node 2** | 2 | 242 | 241 | 1237 | 419 | 2139 |
| **Node 3** | 4 | 484 | 484 | 2199 | 1413 | 4580 |
| **Node 4** | 3 | 363 | 363 | 1855 | 686 | 3267 |
| **Node 5** | 2 | 242 | 242 | 1141 | 365 | 1990 |
| **Node 6** | 3 | 363 | 363 | 1770 | 667 | 3163 |
| **Total/Type** | 16 | 1936 | 1934 | 9406 | 4084 | 17360 |
| **Avg/Node** | 2.7 | 322.7 | 322.3 | 1567.7 | 680.7 | 2893 |
| **Avg/Link** | 1 | 121 | 120.9 | 587.9 | 255.3 | 1085 |
| **Avg/Link/Min** | | 1 | 1 | 4.9 | 2.1 | 9.04 |

# 5 Related Work

Very limited work has been done in the routing area of NDN. We previously developed OSPFN [14], an extension of OSPF (Open Shortest Path First) for routing in NDN, and deployed this routing extension in the NDN research test bed. OSPFN defines a new type of Opaque LSA to carry name prefixes in routing messages. It installs the best next-hop to each name prefix in the FIB, and additionally operators may manually configure a list of alternative next-hops for OSPFN to install in the FIB. Although OSPFN can build a FIB with name prefixes, it has significant limitations. As with conventional IP routing protocols, OSPFN still uses IP addresses as router IDs, relies on GRE tunnels to cross legacy networks, and computes only a single best next-hop for each name prefix. Experience from OSPFN deployment suggests that managing IP addresses and tunnels are major operational problems, and inadequate multipath support limits NDN's effectiveness.

The routing protocol proposed by Dai et al. [2] is similar to NLSR on the surface, but it differs from NLSR in the following aspects: Firstly, it uses OSPF to collect the topology and compute shortest path, whereas NLSR uses ChronoSync to disseminate routing updates. Secondly, their routing message exchanges do not follow basic NDN philosophy, and therefore cannot exploit advantages offered by NDN like built-in security. Finally, their multipath forwarding is limited to content served by multiple producers.

Torres et al proposed a Controller-based Routing Scheme (CRoS) for NDN [12]. The controllers store the network topology, calculate the routes, and store named

data locations so that they can install a route for any named data in the network. The idea of having decentralized controllers is very intriguing, however the network needs to be flooded with specially formatted Interest messages to search for controllers, which can ultimately lead to high routing message overhead.

In [11], the authors proposed a Cooperative Routing Protocol, which focuses on a FIB reconstruction based on the content retrieval statistics of a router. In this scheme, the authors define the network as a set of requester routers (generating or receiving interests and forwarding them to next-hop routers if the content is not available in the Content Store) and provider routers (advertising name prefixes, i.e. connected to producers). Based on the content retrieval statistics, one router reconstructs FIB entries to aggregate multiple flows of interest for similar content. Although the proposed scheme improves network performance, it uses content retrieval statistics to reconstruct FIB, which would not be possible if the FIB was not constructed by some other routing protocol first. In [3], the authors proposed the name-based routing scheme NetInf, which adopts a hierarchical Distributed Hash Table (DHT for name-based routing. The authors proposed to have DHTs for each Point of Presence (PoP) for name resolution. These PoP-level DHTs are aggregated into a higher-level DHT for the resolution of names in a larger domain. The topmost DHT in the NetInf hierarchy (known as REX) stores indices for all the content in the network. This huge scale of indices storing for each content in the topmost DHT creates a network performance bottleneck.

# 6 Future Work

NLSR functions in a single authoritative area/domain. With the growth of NDN, it will also need to mature to function in multiple domains. This work is the first step towards the development of an inter-domain routing protocol for NDN. Moreover, this current simple implementation of multipath calculation results in a linear increase in CPU processing time with the number of neighboring nodes. We will investigate this multipath calculation algorithm to improve the performance of NLSR.

NLSR uses ChronoSync for LSA dissemination, which effectively distributes the names of the updates across the network, however NLSR still needs to send Interests throughout the network to fetch these updates. But if the updated data could piggy-back on the sync data exchanges (which contain only names), then NLSR would not have to send those extra Interest messages, resulting in decreased message exchanges, reduced network convergence time, and improved routing efficiency.

The trust model functions properly under the assumption that NLSR will be within a single authoritative domain where a single entity is the trust anchor. But in multiple authoritative domains, prior assumption will no longer stand, and this trust model then will need to find an effective way to determine a trust anchor and derive trust, which is also a future goal of this NDN routing research.

# 7 Conclusion

The design of Link State Routing protocol is a long studied, well-understood subject for IP. However, devising an efficient Link State Routing protocol for NDN has proved to be an interesting challenge, and our design of NLSR so far has served as a great learning experience. NLSR departs from the conventional IP-based routing protocol's single path forwarding with multiple path forwarding options, and it also propagates name reachability to meet NDN's routing needs. Our key gains from this experience, have come from NLSR specifically needing to develop a new application on top of NDN, which requires a systemic name space design, careful design of a trust model for key authentication, and most importantly, thinking through adjustments to NDN's design pattern. The results represent the first step of our endeavor to explore new routing schemes and extend into inter-domain routing for NDN.

# Bibliography


[1] C. Bian, Z. Zhu, E. Uzun, and L. Zhang. Deploying key management on NDN testbed. Technical Report NDN-0009, February 2013.

[2] H. Dai, J. Lu, Y. Wang, and B. Liu. A two-layer intra-domain routing scheme for Named Data Networking. *Globecom 2012 - Next Generation Networking and Internet Symposium*, December 2012.

[3] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl. Network of information (netinf) - an information-centric networking architecture. *Comput. Commun.*, 36(7):721–735, April 2013.

[4] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard. Voccn: Voice-over content-centric networks. In *Proceedings of ACM*, 2009.

[5] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proceedings of ACM CoNEXT*, 2009.

[6] L. Zhang et al. Named data networking (NDN) project. Technical Report NDN-0001, PARC, October 2010.

[7] J. Moy. OSPF version 2. *RFC 2328*, Apr. 1998.

[8] D. Oran. Osi is-is intra-domain routing protocol. *RFC 1142*, Feb. 1990.

[9] F. Papadopoulos, D. Krioukov, M. BoguÃśÃą, and A. Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *Proceedings of INFOCOMM*, 2010.

[10] PARC. CCNx open source platform. `http://www.ccnx.org`.

[11] S. Tarnoi, K. Suksomboon, W. Kumwilaisak, and Y. Ji. Cooperative routing protocol for content-centric networking. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 699–702, October 2013.

[12] J. Torres, L. Ferraz, and O. Duarte. Controller-based routing scheme for

Named Data Network. Technical report, Electrical Engineering Program, COPPE/UFRJ, December 2012.

[13] University Of Arizona. ccnping. `https://github.com/NDN-Routing/ccnping`.

[14] L. Wang, A. M. Hoque, C. Yi, A. Alyyan, and B. Zhang. OSPFN: An OSPF based routing protocol for Named Data Networking. Technical Report NDN-0003, July 2012.

[15] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang. Adaptive forwarding in named data networking. *SIGCOMM Comput. Commun. Rev.*, 42(3):62–67, June 2012.

[16] Z. Zhu and A. Afanasyev. Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking. In *Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP 2013)*, Goettingen, Germany, October 2013.